

AFRL-IF-RS-TR-1998-233
Final Technical Report
January 1999



ALGEBRAIC AND TOPOLOGICAL STRUCTURE OF QOS (END TO END) WITHIN LARGE SCALE DISTRIBUTED INFORMATION SYSTEMS

Mitchell J. Anderson, Consultant

Mitchell J. Anderson and Robert Mathews

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

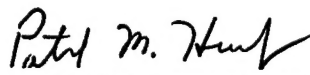
**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

1 9 9 9 0 2 1 7 0 3 9


This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1998-233 has been reviewed and is approved for publication.

APPROVED:


PATRICK M. HURLEY
Project Engineer

FOR THE DIRECTOR:


WARREN H. DEBANY JR.
Technical Advisor, Information Grid Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFGA, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE January 1999		3. REPORT TYPE AND DATES COVERED Final Jan 98 - Jul 98
4. TITLE AND SUBTITLE ALGEBRAIC AND TOPOLOGICAL STRUCTURE OF QOS (END TO END) WITHIN LARGE SCALE DISTRIBUTED INFORMATION SYSTEMS			5. FUNDING NUMBERS C - F30602-98-C-0035 PE - 62702F PR - ALGI TA - TO WU - P1	
6. AUTHOR(S) Mitchell J. Anderson and Robert Mathews			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Mitchell J. Anderson, Consultant HCR - Box 5692 Keaau HI 96749			10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-1998-233	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGA 525 Brooks Road Rome NY 13441-4505				
11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Patrick M. Hurley/IFGA/(315) 330-3624				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) End-to-end quality of service (QoSete) in large scale distributed information systems (DIS) is essential for intelligent system acquisition and design. Without well-developed mechanisms to measure system performance in terms of user requirements, systems can only be built in an ad-hoc manner. The QoSete model formalizes the identification of user requirements and provides performance metrics that are based on empirically measurable attributes that indicate how well user requirements have been met. These performance metrics manifest through the Benefit Function, which is then used for resource dimensioning. Ideally, large scale DIS should be designed and built to efficiently match both system and capital resources to pre-defined user requirements. This avoids users receiving unacceptably low levels of service and/or expending unnecessary capital. The current barriers to achieving QoSete include a lack of understanding of user requirements, the lack of a practical interface between the users and system designers, the lack of a common framework for integration of concepts between the various areas of expertise within the system, and the lack of a mature composability theory that allows such systems to be designed in a modular sense. The work herein addresses to some degree (primarily at a high-level) each of these obstacles, as well as providing insight into a number of related issues.				
14. SUBJECT TERMS Distributed Information Systems (DIS), Quality of Service (QoS)			15. NUMBER OF PAGES 44	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Acknowledgments

The authors would like to acknowledge the assistance provided by both Patrick Hurley and Thomas Lawrence, of AFRL/IF, who provided support in the form of much needed and stimulating brainstorming sessions.

CONTENTS

EXECUTIVE SUMMARY	iv
1.0 INTRODUCTION	1
2.0 USER REQUIREMENTS	1
2.1 Responsiveness	2
2.2 Interoperability	2
2.3 Coverage and Connectivity	3
2.4 Mobility	3
2.5 Human/Environmental Interaction	3
2.6 Measurability	3
2.7 Affordability	4
2.8 Structural Rules and Conventions	4
2.9 Flexibility	4
2.10 Security	4
2.11 Survivability	5
2.12 Summary	5
3.0 THE LAWRENCE QoS MODEL	5
4.0 THE ROLE OF TPA IN DELIVERING QoS_{ete}	6
5.0 RESEARCH RESULTS	8
5.1 Approach	8
5.2 The Benefit Function	9
5.3 QoS _{ete} – A posteriori Definition	10
5.4 QoS _{ete} – A priori Definition & the Utility Function	11
5.5 Application/System Decomposition	12
5.6 Composability	14
5.7 Algebraic Structure	15
5.8 Resource Management	17
5.9 Topological Structure	19
5.10 Statistical Considerations	21
5.11 Logistical Considerations	22

5.12	Global Perspective	23
5.12.1	Global Benefit Function	23
5.12.2	Global Utility Function	24
5.12.3	Global Composability/Algebraic Structure	24
5.12.4	Global Adaptive Resource Management	25
5.13	SUMMARY	26
5.14	FUTURE DIRECTIONS	26
	REFERENCES	27
	Notation	29

Figures

1	The Role of TPA – Equating System Capabilities to User Requirements ..	8
2	The Benefit Function	10
3	The Utility Function	12
4	Multi-Level Resource Management	19

Executive Summary

End-to-end quality of service (QoS_{ete}) in large scale distributed information systems (DIS) is essential for intelligent system acquisition and design. Without well-developed mechanisms to measure system performance in terms of user requirements, systems can only be built in an ad-hoc manner. The QoS_{ete} model formalizes the identification of user requirements and provides performance metrics that are based on empirically measurable attributes that indicate how well user requirements have been met. QoS_{ete} is a special case of QoS in the sense that QoS metrics have values for each component/subsystem in a system, while QoS_{ete} metrics specifically refer to **end-to-end** system performance. In this sense QoS is more general and can be applied at both the system and subsystem level. Ideally, QoS_{ete} metrics define end-to-end performance in terms of the component QoS metrics. QoS_{ete} performance metrics manifest through the Benefit Function, which is then used for resource dimensioning. Ideally, large scale DIS should be designed and built to efficiently match both system and capital resources to pre-defined user requirements. This avoids users receiving unacceptably low levels of service and/or expending unnecessary capital. The current barriers to achieving QoS_{ete} include a lack of understanding of user requirements, the lack of a practical interface between the users and system designers, the lack of a common framework for integration of concepts between the various areas of expertise within the system, and the lack of a mature composability theory that allows such systems to be designed in a modular sense. The work herein addresses to some degree (primarily at a high-level) each of these obstacles, as well as providing insight into a number of related issues.

This report presents a conceptual view of how the Lawrence QoS model can be used as an appropriate point of departure for providing a common framework for equating system/subsystem capabilities to user requirements. This is achieved by first presenting a summary of user requirements, relating whenever possible these requirements to the QoS_{ete} model. Following the section on user requirements is an outline of the Lawrence QoS model and a discussion of the precise role it plays in the delivery of QoS_{ete}. Such a role is heavily dependent on the development of the Benefit Function and related mechanisms. This 'individual' Benefit Function exists for each user/application/time and is used to quantify the subjective benefit to the user executing an application at a particular point in time.

Two distinct, yet closely related, perspectives were identified from which to address QoS_{ete}. In the passive, a posteriori (past tense) perspective, QoS_{ete} is simply the benefit perceived by the user from an already executed application. This model of QoS_{ete} is used to gain insight into the more dynamic processes required for adaptive resource management. The Utility Function is then developed to map system resources to application tasks, using a decomposition of applications and systems/sub-systems into logical tasks and the corresponding system components used to execute these tasks. The

Utility Function composed with the Benefit Function yields the QoS_{ete} mapping of system resources, relative to each application, to user benefit and hence provides a measurement of system performance in terms of satisfying user requirements.

The decomposition of applications into logical tasks, together with the subsequent mapping to system resources via the Utility Function, provides a basis for describing QoS_{ete} in terms of the QoS delivered by each component. This description takes form in the section on Composability and concludes with a presentation on Algebraic Structure. Each QoS_{ete} attribute requires identification of its aggregate values in terms of its component values. An introduction into this process is presented here, with preliminary results provided for the class-level attributes. Further identification depends first on the identification of each QoS_{ete} attribute through the decomposition of the high-level attribute classes.

A model for adaptive resource management is then presented which outlines a strategy for multi-level resource management. This model requires that application tasks be further grouped into sub-sequences of tasks, with each sub-sequence under the control of a single resource manager. Requests for required levels of performance (given in terms of the QoS_{ete} attributes) travel from higher level managers down, with possible execution "paths" based on available resources traveling back up. A mathematical, topological structure for the classification and comparison of such paths is then presented, along with statistical and logistical considerations.

The final section addresses the difference between QoS_{ete} for the individual user and QoS_{ete} for an aggregate community of users. Just as each individual user perceives a subjective benefit from each execution of an application, so too does a community of users perceive a subjective benefit from the aggregate execution of a set of applications at each point in time. This global level of analysis extends to the Utility Function and to adaptive Resource Management as well. Strategies for identifying overall benefit to a user community in terms of individual benefit are presented. This leads to a definition for Global QoS_{ete} and a discussion of tradeoffs on a global scale.

Complex DIS are very costly and inefficient unless they can effectively satisfy very precise user requirements. These requirements are becoming much more demanding, both in terms of the sophistication of the applications the user community employs to complete its business, and the sheer magnitude of the user community itself. The next generation of users requires information systems to be able to prioritize both users and applications, and to dynamically re-allocate resources to provide assurances that the highest priority jobs receive the highest probability of completion, while at the same time allowing lower priority jobs to gracefully shut down. Such decisions, addressing the tradeoff between fixed resources, can be made intelligently only if quantifiable measurements of the effects due to such actions can be [pre] determined. This is the role

that QoS_{etc} plays, providing a metrics-based methodology and theory for designing and implementing complex DIS and for measuring the success thereof.

1.0 Introduction

This Final Technical Report summarizes the work performed under Contract No. F30602-98-C-0035, between Mitchell J. Anderson, Consulting, and AFRL/IF from Jan. 28, 1998 through August 28, 1998; and is designated as CLIN 0003, as stipulated in section B of the contract. Patrick Hurley, AFRL/IFGA, is the program manager for this contract.

The primary goals of this project are to identify user/user-class requirements within complex DIS, the relationship between these requirements and end-to-end Quality of Service (QoS_{ete}), and a mathematical foundation and structure for such QoS_{ete} in complex distributed information systems (DIS). For a comprehensive survey of QoS Architectures, together with an extensive bibliography and highlights of many current efforts, approaches, and concerns regarding QoS, see [CA96]. This particular research effort was designed to develop a high-level approach to mathematically describing the quality of service model, and takes as a point of departure the Lawrence QoS Model, introduced by Thomas Lawrence in [L97] and since refined and developed in [SCDSL97], [CSSDL97], and [WNCHL97], among others.

This report proceeds by first outlining the user requirements identified as most relevant to the quantification and mathematical development of QoS theory, together with an overview of the Lawrence QoS Model and its role in delivering QoS_{ete}. A presentation of the [user] Benefit function, as well as two perspectives of QoS_{ete} follows this general overview. The passive, a posteriori (past-tense) perspective is descriptive in nature, simply recording the level of QoS_{ete} delivered to the user, and forms a foundation for the second perspective. The active, a priori (future) perspective is based on the decomposition of both applications and system resources and the corresponding mapping between the two, and takes into account the statistical nature of the QoS attributes. The a priori perspective leads to the definition of the Utility Function, which maps system resources to QoS_{ete} attributes, and subsequently to the development of a multi-level Resource Management methodology. A rudimentary composability theory is presented that describes end-to-end system QoS in terms of sub-system QoS, followed by an algebraic structure that is closely related to composability. Following a section addressing the topological structure of resource management (from a distance/metric/comparison-based perspective) the report concludes with a presentation of a global perspective to both QoS_{ete} and resource management.

2.0 User Requirements

The primary goal of QoS_{ete} is to satisfy end-user requirements, and the primary goal of this research is to develop the mathematical foundation for the structure upon which QoS_{ete} is based. In order for the resulting mathematical foundation to be relevant, it must be flexible enough to incorporate as many of the desirable user requirements as possible.

It must also facilitate the measurement of the effectiveness or level of satisfaction (from a user perspective) relative to the delivery of such requirements. Therefore, the first step in implementing QoS_{etc} within large scale DIS is identifying user requirements. Rather than providing an exhaustive survey of such requirements, the authors present a summary of their findings directly below. The goal here was to capture the essence of large scale DIS as a system of sub-systems in order to enable the subsequent building of the mathematical structure to support QoS_{etc} within such systems. The authors identified the primary user requirements and classified them into eleven not necessarily mutually exclusive areas including responsiveness, interoperability, coverage and connectivity, mobility, human/environmental interaction, measurability, affordability, structural rules and conventions, flexibility, security, and survivability. These user requirements were catalogued through meetings with key user communities including government (tri-service, JS, JBC, DARPA, NIST, PACOM), academia (Syracuse, GMU, Penn State, LSU) and industry (Bell Atlantic, HP, Sun Microsystems, AT&T, Cabletron, MITRE, BBN).

2.1 Responsiveness

Responsiveness is closely related to QoS_{etc} in the sense that it addresses the system's ability to respond or to deliver information in a manner that is acceptable to the user. For example, users expect the system to respond in a timely manner, to deliver the right amount of data with little or no variance in either arrival time or the quantity of data delivered, and to be free from errors. In order to maximize system performance, this QoS_{etc} must be delivered in an efficient manner. Thus, since system resources will always be finite and hence limited, systems must develop fair preemptive capabilities. They must have multi-level preemption capabilities to handle national security issues as well as assuring that the highest priority tasks correspond to those with the highest probability of completion. Service level agreements between the user and the system, designed with an understanding from both sides of what is desired and what can be 'guaranteed', must be developed. If these agreements are coupled with user friendly feedback from the network regarding demand and available resources, the user could possibly become proactive in helping to increase responsiveness on a real time basis.

2.2 Interoperability

Users are primarily interested in the interoperability of the information itself, not with that of the equipment. They want to be able to integrate information from multiple distributed sources, a capability sometimes referred to as All Source Fusibility. Information today, especially within C2 operations, can arrive quickly from a variety of sources, and in a variety of formats. Objects and/or functions from one application need to be able to transport, in a modular manner to other applications. Further, information that is provided at one platform must be consistent with that provided at other platforms. This sort of information portability is evidenced today in an introductory sense through middleware products such as CORBA, which assures that data precision arrives in a consistent manner. Future applications will require much more flexibility than the static

precision offered/required by CORBA. A well-defined and developed QoS_{etc} theory will assist in this process by providing a common framework and lexicon from which to work.

2.3 Coverage and Connectivity

With respect to coverage and connectivity, users want access to information on a global scale, available in the remotest regions. This is true whether or not the user is from the military or civilian population. This ubiquitous coverage and connectivity should include both push and pull capabilities and should be re-configurable in real time in order to facilitate the dynamic nature of information operations. The implications to QoS_{etc} here are that regardless of the location, the user requires access to information while consistently maintaining high levels of QoS_{etc}.

2.4 Mobility

Users are particularly interested in access to information on a mobile basis, the transition from static to mobile being transparent in nature. The philosophy here is that the user and not the computer is the end component in the system. Satellite technology will introduce bandwidth and other restrictions, making it necessary to be much more aware of the need to match functionality to system resources. Thus, a robust methodology for resource management is a necessity with respect to mobility.

2.5 Human/Environmental Interaction

This area of requirements is related to the users' desire for user-friendly systems. Users want systems in which they can quickly learn the basics, utilizing such features as 'just in time' learning. This area introduces QoS_{etc} from a utility perspective, but remains relevant with respect to the usual QoS_{etc} attributes of Timeliness, Precision, and Accuracy (TPA). In particular, users want to be provided just the right amount of information, not so much as to be overwhelmed, but not so little as to be unable to successfully complete the mission at hand. This relates directly to the precision attribute in TPA. In this same light, users want a set of core functions that cross different applications, so that they are not forced to repeatedly re-learn new functions. Further, users want the system to be intelligent in the sense that it should know each user's profile so that it can respond to the individual user's needs automatically.

2.6 Measurability

Measurability is a technical requirement from the user, stemming from the fact that advanced capabilities with respect to QoS_{etc} cannot be delivered otherwise. Further, from the users' perspective, if more feedback was available to the user regarding excessive system load and anticipated low-level performance, the user could intelligently respond by re-apportioning application loads, shutting down the unnecessary applications to better ensure the successful completion of the more important functions. This goes hand in hand with the type of resource management discussed in detail below, and speaks to the need for prioritization, preempt-ability, and paying for service, all of which are

interrelated. One of the ultimate goals of QoS_{etc} is to provide an idea of what constitutes 'guaranteed' service and how one might go about the necessary negotiation.

2.7 Affordability

In effect, affordability is tied directly to the technical problem of integration. The user wants to utilize COTS and GOTS products because they are inexpensive and they work very well when limited to the jobs for which they were originally designed. With respect to QoS_{etc}, such integration within heterogeneous systems can be affordably achieved only with a clear understanding of the resulting system performance, where performance is defined in terms of user requirements. One way to keep the costs associated with system evolution down is to involve the user whenever possible, particularly allowing the user to extend the life of applications by facilitating the ability to contribute to application design and modification.

2.8 Structural Rules and Conventions

In order for systems to function more effectively, it is imperative that both the user and the provider know what is and is not allowed. Thus, as QoS_{etc} and the associated resource management mechanisms are developed, associated rules as well as enforcement mechanisms also need to be developed. To assist this process, the system needs to include the necessary capabilities to develop audit trails that are designed around the QoS_{etc} attributes, and finally to provide automatic management capabilities to deal with the inherent instability associated with the complexity of distributed systems.

2.9 Flexibility

Flexibility again addresses the user-friendliness of the system, but from a more dynamic perspective. Users want upgrade-ability, as well as backward compatibility. They want systems that are designed in a modular manner, allowing them to scale as necessary as well as making them easy to design, repair, and upgrade. They want things simple. They want nomadic computing and communications to be transparent, as well as QoS_{etc} requirements such as negotiation of services. Further, they want new applications and services to appear in a much more fluid manner as opposed to new capabilities suddenly appearing, requiring the user to quickly adjust mid-stream. If QoS_{etc} mechanisms are to be introduced successfully they will need to take into account these user requirements.

2.10 Security

Users want assurances that information received has not been maliciously altered in any manner during transit. They also want authentication and non-repudiation 'guarantees' that indicate whether or not information received originated from the intended source, and whether or not information sent will be accepted. Users also want feedback verifying that desired events actually occurred. Authentication and non-repudiation need to be multi-level, time synchronized, and role based. Users need to be able to ensure that information is made available only to those for which it was intended. These issues are closely related to all three classes of QoS_{etc} attributes described below, given that it

appears that each property can be delivered through the appropriate manipulation of one or more attribute.

2.11 Survivability

This user requirement speaks directly to the issue of QoS_{etc} in the face of anomalies. Users want information systems to continue providing services despite adverse conditions. They want to be able to demand more and more from systems. When systems fail to keep up, they want them to respond in a user-friendly manner, gracefully degrading services, allowing less important functions the time necessary to shut down, shifting limited resources to the more important applications. Assurances of survivability should be given through user-system agreements, with each set of user/user-class requirements contained in user profiles. These are precisely the types of QoS_{etc} issues that the resource management mechanisms need to address. Users would also like feedback mechanisms so that they can proactively take part in the management process.

2.12 Summary

The user requirements presented above include a number of recurring themes, most of which are directly related to the QoS_{etc} model. The user community wants systems to be receptive to their needs; they want quality of service capabilities, better utilization of limited resources, and active participation. They want the mechanisms enabling these capabilities to be transparent to the user. In order to achieve these goals, much more dynamic information needs to be made available to the appropriate party(s), either to the user, the resource manager, or both. Given the fact that systems will continue to grow, and that the complexity of applications will continue to increase the load on systems that are already pushed to their limit, it becomes increasingly important to develop mechanisms that will more efficiently match user demands to system capabilities, or vice-versa. Preemption mechanisms need to be developed to enable those functions that are most critical to mission success to have the highest probability of completion. The remainder of this report illustrates that these user requirements correspond very closely to the goals of QoS_{etc} .

3.0 The Lawrence QoS Model

The Lawrence QoS Model states that the different levels of service provided to individual users/user-classes through the execution of applications within complex distributed information systems can be completely described in terms of the three quantifiable QoS attributes, Timeliness (T), Precision (P), and Accuracy (A). Timeliness in this case describes when data is produced, how long is required to transport data from point A to point B, the amount of variance in the data arrival times, etc. Precision describes the quantity of data produced or transported and is directly related to the amount of detail delivered by applications. Accuracy refers to the bit error associated with the data produced or transported. The hypothesis of the Lawrence QoS Model is that each user requirement falls within one of these three classes of QoS attributes. One essential task

not addressed within this research, therefore, is to identify the decomposition of the three classes relative to the set of user requirements.

One assumption of the Lawrence QoS Model is that two executions of the same application can receive different levels of QoS_{etc} only if they differ in at least one aspect of their timeliness, precision, or accuracy. For example, one might provide more detail (i.e. a higher level of precision), less bit errors (a higher level of accuracy), or less latency (a higher level of timeliness). The level of QoS_{etc} a user receives from a particular execution of an application is both subjective and objective. It is subjective in the sense that each desired or perceived level of QoS_{etc} depends on the particular user and may change over time. It is objective in the sense that the values for T, P, and A can be empirically measured, independent of the user, the time of execution, or the resources used to deliver the application. In effect, these attributes are mathematically metrizable. The Lawrence QoS model is based on the following three principles:

1. QoS_{etc} is meaningless unless the values of all three attributes are known or specified.
2. Assuming a system running an application has no reserve resources, the only way to increase the performance of one of the attributes for the application, without affecting another application running on the same system, is to decrease the performance of at least one of the other attributes for the application.
3. Assuming a system running an application has no reserve resources, the only way to increase the performance of one of the attributes for the application is to decrease the performance of at least one of the attributes for an application running on the system.

Note that the second principle is a local (to the application) principle, while the third principle is global to the system.

The QoS TPA attributes are both fuzzy and statistical. They are fuzzy in the sense that multiple values and/or combinations of the values T, P, and A, corresponding to the execution of a particular application at a fixed point in time, may provide equal benefit to the user; and they are statistical in the sense that each value of T, P, and A, corresponding to a particular application/HW/SW/anomaly configuration, is given by a probability distribution.

4.0 The role of TPA in Delivering QoS_{etc}

The ultimate goal of quality of service is to maximize resources from a user perspective. This implies two necessary capabilities; 1) understanding in a quantifiable manner user requirements so that 'maximize' is well defined (represented by the upper-level mapping in Figure 1), and 2) mapping system resources to those requirements (represented by the lower-level mapping in Figure 1). Unfortunately, system capabilities are usually given in terms of Processing (P), Communications (C), and Data Management (D), while user

requirements are generally much more broad, including properties such as functionality, reliability, security, etc.

Historically, the user community in general has difficulty **articulating** end-to-end performance requirements within complex DIS. Nevertheless, given two applications running simultaneously, individual users can usually determine, albeit subjectively, which is performing more to their liking based on some basic properties. For example, one application output displayed on a monitor may appear to have more clarity, more intensity of color, more detail, less distortion, or less jitter than another, and hence may appear to provide more information and is therefore perceived as more desirable. Unfortunately, the system capabilities are given for components only, if at all, and not for end-to-end systems; and, these capabilities are given in terms such as clock speed, bandwidth, instructions/sec., etc. What is required is a conceptual, unifying interface between the user and the system to bridge the gap between these two contrasting lexicons. The Lawrence QoS Model provides this interface.

Each of T, P, and A forms a class of quantifiable QoS attributes that together play a role that is comparable to the role the prime numbers play in number theory where every integer can be "broken down" and represented in terms of prime numbers. It is the hypothesis of the Lawrence Model that all user requirements can be decomposed and represented in terms of the elements of T, P, or A. Thus, corresponding to each user, application, and point in time, there is a set of empirically measurable TPA values that describes the user's requirements at time t (represented as the TPA set to the left of the middle layer in Figure 1). Similarly, system capabilities can theoretically be mapped to the same set of attributes (represented as the TPA set to the right of the middle layer in

Figure 1). If these two sets intersect, then it is possible to meet user requirements with available resources, thus completing the QoS_{etc} mapping to the left. If not, then QoS mechanisms such as tradeoffs between the QoS attributes, available resources, and/or applications must be implemented. In any case, the Lawrence QoS Model plays an important role in better understanding system performance and capabilities.

The Role of TPA – Equating System Capabilities to User Requirements

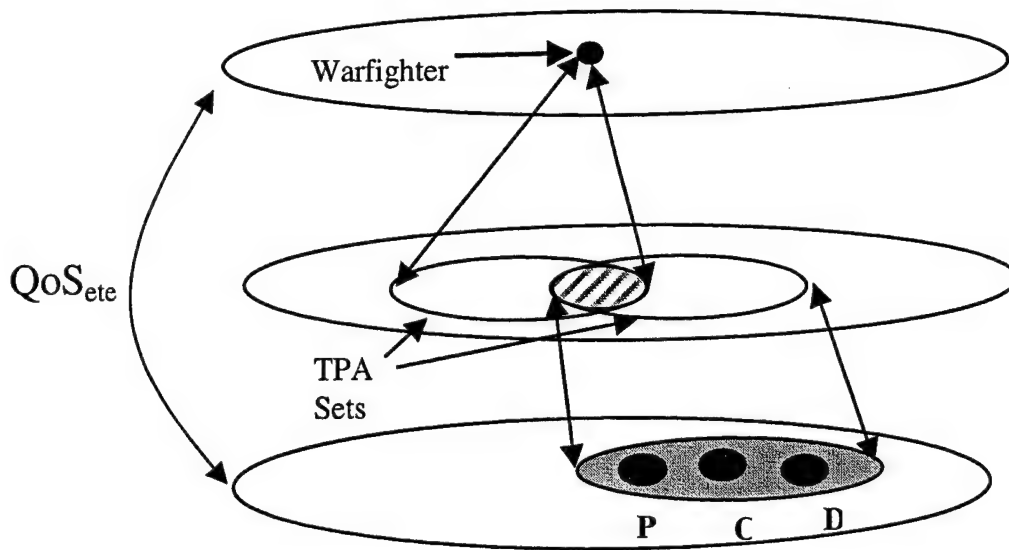


Figure 1

The Lawrence QoS Model, while clearly not a complete solution to the QoS_{etc} problem, provides a very promising structure for addressing the problem of matching information system resources to user requirements. It provides a common framework for communication and understanding between the different areas of expertise within the system such as middleware, network management, database management, and applications development. From the engineering perspective, TPA provides a common metric for comparing the performance of two different systems running the same application for the same set of users.

5.0 Research Results

5.1 Approach

The approach the research followed was to first develop appropriate mathematical definitions for the QoS_{etc} - related objects, and then, based on these definitions, to provide heuristic topological and algebraic structure to QoS_{etc} . The research identified two perspectives from which to address the objects within QoS_{etc} . The passive perspective represents an a posteriori (after-the-fact) view of QoS_{etc} , while the active perspective represents an a priori (yet to be executed) view. For example, from the a posteriori perspective, once an application has completed execution, a particular level of service is completely determined by the resulting empirically observable/measurable values for

each of the QoS_{ete} attributes, together with the subjective benefit having been perceived by the user at that time. On the other hand, the a priori perspective addresses the problem of delivering desired user requirements based on the probability that the system can deliver particular TPA values and is thus given statistically as a distribution. As such, QoS_{ete} in the a posteriori case constitutes a particular mathematical object different than, albeit related to, that in the a priori case.

The a posteriori perspective provides objective information (quantified values for TPA) acquired from prior application executions that can be used to provide insight towards both system design and resource management. In particular, for example, a single a posteriori value for latency (a timeliness attribute), corresponding to a single execution of an application or sub-application task, contributes to the a priori statistical representation for latency for the same application executing under similar conditions. It is important to note that the objective TPA values themselves are independent of the subjective level of service desired by the user. The a priori perspective plays a significant role with regards to resource management.

The mathematical results of the research are presented below in the following order. First the Benefit function is developed from an object definition view. The Benefit function plays a significant role in translating objective TPA values to subjective user benefit. The QoS_{ete} object is then defined, first from an a posteriori perspective and then from an a priori perspective. The a priori perspective uses the development of the Utility function which describes all possible mappings from available system resource to TPA sets. Once these objects have been defined, along with a methodology for decomposing applications and system resources relative to application tasks, the mathematical results are presented in the sections on composability and algebraic structure. The Resource Management object is then defined, followed by a corresponding topological structure thereof, as well as statistical and logistical considerations/observations. The final section presents a global perspective of Benefit, QoS_{ete} , and Resource Management.

5.2 The Benefit Function

The first step in the process of defining QoS_{ete} after the extraction of requirements from the individual user or group of users, is the subsequent translation, via for example an interface mechanism, of those requirements into the TPA framework. With this in mind, let D denote the set of all possible TPA triples for a particular application running on a system at a fixed time, t . Note: as long as each of T , P , and A is well defined for the application, then D is a well defined set. For example, if T represents latency (the difference in time from data input to output), P represents the volume of data output, and A represents the bit error rate of the data between input and output, then each TPA triple is either empirically possible or it is not possible. As illustrated in Figure 2, each element of D provides a subjective "benefit" to the user (the height on the surface, corresponding to each point in D). Thus, associated with each user, application, and time t , is a benefit

function $B_{U,App,t}: D \rightarrow [0,1]$ which maps each quantified element of D to a subjective benefit, scaled between 0 and 1.

Clearly, $B_{U,App,t}$ is not one-to-one since it is highly probable that multiple values of TPA are considered of equal benefit to the user. For example, data that arrives quickly but with less clarity (i.e. high T , low P) might be of equal benefit as the same data that arrives a little later but with better clarity (i.e. lower T , higher P). In fact, this is the basis for making intelligent tradeoff decisions between system resources. $B_{U,App,t}^{-1}[b_1, b_2]$ denotes the pre-image of the benefit interval $[b_1, b_2]$ and hence represents all TPA triples that user U running application App at time t perceives as providing a benefit value between b_1 and b_2 . In particular, if the user classifies all benefits above a fixed value, say b , to be acceptable, then $B_{U,App,t}^{-1}[b, 1]$ denotes all acceptable TPA values for user U running App at time t (Figure 2, the un-shaded region in D). Notice that from the system perspective, $B_{U,App,t}^{-1}[b_1, b_2]$ is an objective set of TPA values, independent of the user or the time the application is run, and as such may be considered a “target” for system capabilities.

It should be noted that the benefit to the user, and hence the benefit function, while dependent on time, is independent of the two perspectives, versus a priori.

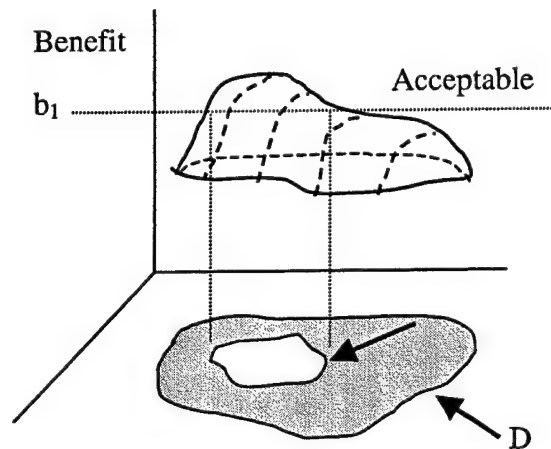


Figure 2

5.3 QoS_{etc} – A posteriori Definition

QoS_{etc} in its simplest form, and from an a posteriori perspective, is a function whose domain is the set of triples (App, S, I) and whose final set is the quality of service space (TPA) representing timeliness, precision, and accuracy. Here App represents a user/application pair, S represents the system configuration (HW/SW/services/anomalies) upon which the application executed, and I is the [past-tense] interval of time during which the application executed. All three components of the domain are necessary since different user/application pairs desire different levels of service, different systems/configurations are able to provide different capabilities, and the information

environment is dynamic. It should be noted that the user/user-class requirements for the same application executed at different times are not necessarily identical. As an ordered triple of quantifiable values the TPA space can be considered a subset of Euclidean 3-space, \mathbf{R}^3 . However, taking into consideration that each of T, P, and A can be decomposed into finer-grain attributes, TPA should more formally be considered a subset of \mathbf{R}^n , for an as yet undetermined positive integer n.

Thus, in this case $QoS_{ete} : V \rightarrow D$, where V is the space of ordered triples, (App, S, I), and D is the space of QoS attributes, (TPA). From a user perspective, QoS_{ete} can be extended further by subsequently sending D into $[0, 1]$ via the benefit function, $B_{U,App,t}$, yielding the following final definition for a posteriori QoS_{ete} :

$$QoS_{ete} : V \rightarrow [0,1].$$

In short, a posteriori QoS_{ete} is the perceived benefit to the user corresponding to a single [past-tense] execution of an application. The a priori definition for QoS_{ete} is much more "rich" from a research perspective and is provided, following the development of the utility function, at the end of the section immediately below.

5.4 QoS_{ete} – A priori Definition & The Utility Function

Given that the benefit function provides a set of acceptable TPA values for user U executing App at time t , namely $B_{U,App,t}^{-1}[b_1, b_2]$, the question becomes "Can the system, S , deliver these objective TPA values at time t ?" Here, S denotes the set of all possible software (SW), hardware (HW), services (sometimes referred to as the bitways and services), and anomaly configurations within a particular distributed system. Towards answering the above question, let S_{App} denote the set of all configurations in S that, if available, could be used to execute App. (The practical representation of S_{App} reflects available system resources and depends on the level of decomposition of both the system components.) Notice that each such execution of App results in an empirically observable, a posteriori TPA value in D delivered by a particular point in S_{App} . For each such point in S_{App} , all possible TPA values (viewed in the a priori sense) taken as a whole provide a statistical representation (rather than a single value) of the resulting TPA. Thus, from an a priori perspective, each configuration/point in S_{App} corresponds to a statistical distribution/representation of possible TPA values in D . For correctness, in the a priori case we consider each point in D to be not an ordinary triple, but such a statistical TPA triple/n-tuple.

At each point in time only a certain percentage of the resources in S_{App} are available to execute App due to the additional applications and services executing at that time, as well as component failures and/or other anomalies. Thus, let $S_{App,t}$ denote the set of resource configurations within S_{App} available to execute App at time t . Further, let $U_{App,t} : S_{App,t} \rightarrow$

D denote the Utility function that associates with each member of $S_{App,t}$ the corresponding possible TPA triple in D . From an a posteriori perspective, only a single point in $U_{App,t}$ represents the actual mapping of resources to the resulting TPA that occurs as a result of executing App at time t . It then follows that this single corresponding TPA output in D , when provided as input for $B_{U,App,t}$, yields end-to-end quality of service for the user at time t . In general, we define $QoS_{ete} = B_{U,App,t} \circ U_{App,t}$. That is, the utility function composed with the benefit function yields end-to-end QoS. This composition maps system capabilities at time t to user benefit, and is captured in Figure 3. From a practical perspective, the user first defines the level of desired benefit. This then defines, through B^{-1} an objective set of TPA values. At the same time, U defines a set of possible TPA values available from the system at time t . If these two TPA sets intersect (Figure 3) then the system can in fact deliver the desired benefit to the user at time t . It should be noted again that in the a priori case, since the range of $U_{App,t}$ is statistical in nature, that QoS_{ete} is statistically based also.

5.5 Application/System Decomposition

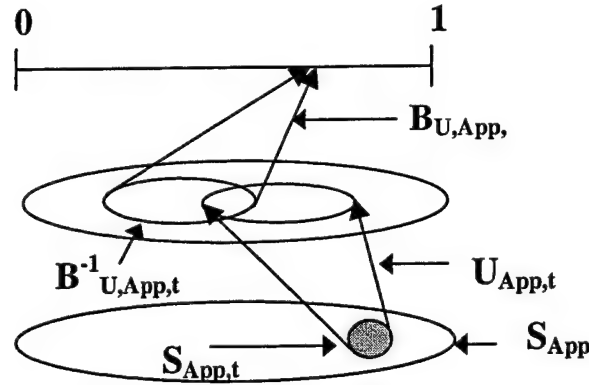


Figure 3

Many of the results presented below (e.g. resource management, composability, and the algebraic and topological results) rely on a decomposition of applications into sub-applications/tasks and/or the system into subsystems/modules. Thus, assume that App can be decomposed into m logical sub-applications, A_1, A_2, \dots, A_m , each executed sequentially, and that $i < m + 1$. For example, App might represent a VTC application necessitating the repeated capturing, processing, and transporting of audio/video frames, or a tracking application that requires regular refreshing of track data. Although we do not provide a rigid and detailed definition for a sub-application here, it suffices to note that each of these 'sub-applications' requires physically distributed resources within the system and that App is realized by the **sequential** execution of the 'sub-applications'. Further, assume that A_i can be decomposed into n logical tasks $\tau = \tau_1, \tau_2, \dots, \tau_n$.

The literature [CSSDL97] refers to such tasks as either *Logical Units of Work* (LuoW), *Logical Realization of Work* (LroW), or *Physical Units of Work* (PuoW), depending upon the granularity or whether or not the concern is logical or physical. It suffices for this research to consider all such tasks as logical units of work. For example, the tasks might be fetch video, fetch audio, package video/audio for transport, transport video from sending LAN to receiving LAN, synchronize audio and video, etc. It is assumed that each task is completed by a specific component of the system, call it S_i , (a PC, the sending or receiving LAN, a component of the telecom, etc.; the granularity is meant to be flexible) and that τ_i is performed during the time interval $[t_i, t_{i+1}]$. It is **not** assumed, however, that these intervals are non-overlapping, since some tasks may be performed in whole or in part simultaneously. In fact, unlike the sub-applications, these n tasks, while listed sequentially, are partially ordered relative to their order of execution by the relation "is equal to or must be completed prior to the beginning of." Since the execution of App is realized through the sequential execution of the A_i , QoS_{etc} for App is completely dependent on QoS_{etc} (the TPA values) for each A_i . Thus, without loss of generality it suffices to investigate A_i alone. Corresponding to the sequence of tasks, τ , is the sequence S of system resources such that τ_j is executed on S_j . This mapping suggests that the fidelity of the decomposition of both A_i into τ and the system into S are inter-dependent. The granularity of the decomposition of A_i depends greatly on the degree to which the system can be decomposed (and understood/controlled) into corresponding resources; the ideal results occur when each task is performed by a simple, well understood, system component. Notice that the union of the members of S in an a posteriori sense represents that point in $S_{App,t}$ corresponding to the execution of A_i .

Next, denote by QoS_j the resulting/expected TPA from the execution of τ_j . That is, each execution of τ_j can be characterized as a transformation of information in the sense that τ_j , for example, requires a certain amount of time to execute (contributing to a change in T), possibly alters the volume of data produced (e.g. via compression or packet loss, contributing to a change in P), and possibly alters the bit error in the data produced (contributing to a change in A). The implication here is that each τ_j plays a role identical to that of App in the above construction of the utility function, with a corresponding similar resulting TPA value. The question before us then is what is the relationship between the sequence of QoS_j corresponding to τ , and the aggregate TPA for A_i , and hence to QoS_{etc} for App? This question is partially answered in the following sections.

5.6 Composability

Utilizing the decomposition of App into sequentially executed sub-applications, A_i , the further decomposition of A_i into partially ordered logical tasks, τ , the mapping of τ to the system resources, S , and the utility mapping, U , which provides a representation of each member of S as a transformation of information, (transforming TPA-in to TPA-out), we provide the following initial hypotheses regarding the composability of these transformations. These hypotheses are intended to be high-level, first approximations to the mathematics of composability. Further granularity will depend on identifying the components of each of T , P , and A (for example, timeliness includes not only latency, but also jitter, variance of jitter, scheduling, etc.), and on obtaining hard experimental data relative to each. The results below have significant implications to resource management (Sec. 5.8).

- Accuracy (bit error rate) is cumulative in the following multiplicative sense.

Let ϵ_j represent the error rate for data during the execution of task τ_j . That is, the probability of data having arrived at the beginning of task j surviving task j is $1 - \epsilon_j$. Thus, assuming accuracy is the percentage of data free from error (which is 1 minus the error rate), and that accuracy is task independent, it follows that the end-to-end accuracy for A_i is given by

$$A = \prod_j (1 - \epsilon_j)$$

which is equivalent to saying the end-to-end error rate is $1 - A$. (It should be noted that included in each ϵ_i is the error resulting from an I/O interface between tasks i and $i+1$.) The implication here is that end-to-end accuracy can be controlled by bounding the error rates of the individual tasks. Simplistically, for example, if the desired bound for end-to-end error is ϵ , then bounding each intermediate error to the n^{th} root of ϵ ensures success. Realistically, since components differ in their levels of reliability and each task differs in its complexity, the component errors can be much less uniformly distributed while still remaining within the end-to-end ϵ error bound. These component error rates, together with those associated with each I/O interface, should result from experimentation.

- Timeliness is additive when restricted to the case of latency.

Let λ_j denote the latency due to τ_j , and let δ_j denote the delay between the completion of task τ_{j-1} and the start of task τ_j , with δ_1 being the scheduling delay prior to the start of τ_1 . Then, if τ is executed sequentially it follows that end-to-end timeliness is given by

$$T = \sum_j \lambda_j + \delta_j$$

In case τ is not executed sequentially, then this sum forms an upper bound for latency and must be reduced by overlapping times in order to achieve equality. Analogous to the accuracy scenario above, an end-to-end latency bound λ can be guaranteed simplistically by requiring $\lambda_j + \delta_j$ to be bounded by λ/n for each j . Again, given the diverse nature of the tasks and the components executing the tasks, more precise (less uniform) bounds can be assigned to individual tasks based on historic, dynamically evolving data.

- Precision (volume of data processed or transported) is determined by the minimum precision throughout execution.

Precision for each task in this case equates to the volume of data output from each task, τ_j . Since the volume of data may decrease from task to task, due to for example compression algorithms during processing or dropped packets during transport, it appears that in the straightforward cases end-to-end precision is given by

$$P = \min_j \rho_j$$

where ρ_j denotes the precision for τ_j . In the case of packet loss, for example, end-to-end loss can again be bounded by bounding the intermediate tasks in an appropriate manner, analogous to the methods above for error rate and latency.

It is important to remember that we have chosen simplistic representations for T, P, and A here, and that each of these attributes contains a number of additional representations with respect to satisfying user requirements, each of which requires an associated theory for composability. In the case of jitter, for example, this timeliness property appears to compose much differently than latency. In fact, since jitter is defined as the variance in latency at the end-user, it doesn't make sense to simply sum the intermediate latencies, nor does it make sense to sum the variances. As is similar with all composability, what is required for the composability of jitter is to unearth the relationship between the intermediate variances and the final variance. Therefore, jitter appears to compose much like the accuracy representation above. This follows because in order to ascertain the resulting variance we must first ascertain the mean, and in order to ascertain the mean, the intermediate distributions must first be summed, which subsequently causes the resulting variance to "spread out", similar to the cumulative multiplicative effect described above.

5.7 Algebraic Structure

Composability provides a foundation for certain algebraic structure within the Lawrence QoS model. The fundamental theory presented here is based on the idea that if task τ_j is executed on resource S_j , transforming the TPA-in into the TPA-out via the information transformation QoS_j , then the sequential execution of τ_j followed by τ_{j+1} will form a

transformation of the information input into S_j and output from S_{j+1} given by the mathematical operation of composition of functions $QoS_{j+1} \circ QoS_j$.

Before proceeding to such a theory we must ensure that the transformations to be composed, QoS_j , have compatible domains and target spaces. This can be achieved by first realizing that when data arrives at S_j , there is a cumulative effect on QoS to that point, say TPA_j^* . Formally, assuming tasks $\tau_0, \tau_1, \dots, \tau_j$ are performed sequentially, in the aggregate they can be considered (informally) a sub-application in the sense that the execution of such a sub-sequence of tasks is the realization of a larger task and as such the composability results from above remain true. Thus, here we augment the QoS domain space defined above, represented by the ordered triples (App, S, I) , to be (App, τ, S, I, TPA^*) . The result is that the domains and target spaces are then equivalent, allowing for mathematical composition, or transformation. That is, the execution of each task depends on the application, its decomposition into τ , available system resources, time, and the TPA (information) values input, and acts as a transformation of that TPA, resulting in TPA-out.

A more general result is true, one that is similar to flow solutions to differential equations. That is, continuing to assume the tasks are executed sequentially, if we denote the transformation represents the execution of tasks τ_i through τ_j , $j > i$, as $QoS_{i,j}$, with the resulting quality of service component of the transformation output denoted by $TPA_{i,j}$, and n is a positive integer, then $QoS_{i,j+n} = QoS_{j,j+n} \circ QoS_{i,j}$, which implies $TPA_{i,j+n} = TPA_{j,j+n} * TPA_{i,j}$. Here, $*$ refers to the cumulative effects of QoS as described in the previous section on composability, and is written from right to left to mimic the mathematical composition of the QoS functions. Notice that $*$ defines an additive structure on the set $\{0, 1, 2, \dots, n\}$ in the sense that $i * j \equiv i + j$ iff $TPA_{0,i} * TPA_{i,i+j} = TPA_{0,j} * TPA_{j,j+i} = TPA_{0,i+j}$, which is true provided $i + j < n + 1$.

Therefore, $*$ resembles a semigroup with identity. Using the composability results above, it follows that the transformation, e , corresponding to the identity is given simply by

$$e = \{(0, \varpi_0, I)\},$$

where 0 represents the transformation requiring no time, I represents the transformation that introduces no bit error, and ϖ_0 can be replaced by any transformation that represents the delivery of an upper bound value for the volume of data (P). Note: composition is defined only for each execution 'flow', since it is defined in terms of $QoS_{i,j}$, which in turn relies on the various sub-systems performing particular tasks, the order and values of which cannot be changed in general. An alternative algebraic structure relative to composition results from considering the sub-application, A_i as a function of time, its realization occurring over the time required to execute its tasks. In this sense, A_i resembles a one-parameter (time) flow, again yielding a similar definition for a

semigroup operation, depending on time, with an associated QoS at time t , namely the cumulative QoS through time t , provided t lies within the time interval of execution.

5.8 Resource Management

For the discussion here we assume the existence of both the benefit function and the utility function defined above. That is, for each triple (U, App, t) there is a function $B_{U,App,t}$ that maps each objective TPA value in the set D of all such TPA values to a value in $[0,1]$; and there is a function $U_{App,t}$ that maps each set of resources available in S_{App} for executing App at time t to a set of TPA values in D . (Figure 3)

To begin the investigation of resource management within a reasonable setting, we restrict our attention to the case where the system in question has a single resource manager, RM, controlling m resources, $s = s_1, s_2, \dots s_m$, (i.e. s is a sequential decomposition of S_{App} into task-oriented resources) and let App be an application attempting to execute on this system during a fixed time interval, I . Further, let App be given logically by the sequence of tasks, $\tau = \tau_1, \dots \tau_n$, which execute sequentially. Heuristically speaking, RM is then a middleware function that maps task requirements to available resources, while attempting at the same time to satisfy user requirements. From an a priori perspective, RM is statistical in nature as indicated above through its relation to the statistically based utility function; and from an a posteriori perspective, RM provides an empirically observable record of resource usage towards application execution. More specifically, RM is a choice function, each choice representing a single point given by the utility function, $U_{App,t}$, and when composed with the benefit function provides a statistical mapping from system resources to user/user-class benefit, that is, QoS_{etc} .

$$RM : (\tau, TPA_p, P_0) \rightarrow (S_{App,t}, D)$$

Here, TPA_p represents the TPA profile of the user/user-class (i.e. those partially ordered levels of QoS desired by the user), and P_0 represents the system policy for delivering requested levels of TPA, depending on system load, anomalies, user-prioritization within the system, etc. Recall that $S_{App,t}$ represents those system resources available at time t to execute App . Note: there are multiple values in the image of RM only because τ represents more than one task, in general, not because of the multiplicity of the TPA values in D , since one of the roles of RM is to choose exactly one value in $S_{App,t}$ for each task, which then leads to a single TPA value (or single statistical distribution) in D . In the case where $m = 1$, that is, where there is only 1 task being performed, RM simply decides which resource under its control will deliver the task.

The user profile, TPA_p , provides user TPA preferences to RM, facilitating intelligent, decision making by RM. As such, TPA_p must contain all information contained in the

benefit function for the user at time t . Although it is possible that the user will desire the same benefit function for every execution of App, RM is clearly a function of time, t being implicitly included in the user profile. Similarly, the system policy, P_0 , for making resource allocation decisions must contain all information required by the system to determine resource allocation at that point in time. Presumably, system requirements change over time; hence P_0 is also a function of time. Note: for simplicity, if we view RM as a short term decision making process, system policy remains constant and need not be included as a coordinate of the domain.

Consider now the case where tasks must be performed by resources outside the control of a single resource manager. For simplicity, assume App is decomposed into logical tasks τ , grouped into two sequences $\tau_{1,1}, \tau_{1,2}, \dots, \tau_{1,n1}$ and $\tau_{2,1}, \tau_{2,2}, \dots, \tau_{2,n2}$ under the control of resource managers RM_1 and RM_2 , respectively. Further, assume RM is a control mechanism as above, with RM_1 and RM_2 mimicking the role of s . From the perspective of RM, each sequence of tasks is simply a single aggregate task. Thus, by induction, each of RM_1 and RM_2 provides "black box" information to RM regarding the available resources and more specifically the corresponding statistical TPA-values deliverable for each of the two higher level tasks. Given that RM above was responsible for the assignment of n tasks within m resources, there is no reason for restricting the sequence of resource managers, RM_1 and RM_2 to two. In general, we assume RM has m resource managers to consider as resources to execute App.

Similarly, if τ is subdivided into n sub-sequences, each of which must be assigned to one of the m resource managers RM_j , again by induction the problem is reduced to the simple case above where RM must make n execution choices. The only difference here is that a resource manager 'manages' its own resources based on detailed information regarding each resource, while lower level resource managers transmit black-box information to the resource manager one level up. In this manner, multi-level resource management can be developed as "granular" or "course" as necessary. For example, from the highest perspective, RM might control only three or four (different for each App) resource managers, representing say the sending LAN, the telecom, and the receiving LAN, with a database resource acting as the fourth. Similarly, the resource manager in charge of the LAN might control three or four sub-nets, each with its own resource manager, and so on (Figure 4). This decomposition may continue down to the level where each computer has its own resource manager.

The set of resource managers is partially ordered by the relation $RM_i \leq RM_j$ only in case RM_j also manages the resources managed by RM_i . Numerous approaches can be developed to attempt to deliver QoS_{etc} utilizing this structure. For example, requirements can travel down from higher-level resource managers to lower level managers, requesting particular ranges of TPA to be delivered from each 'black-box' resource. Simplistically, for example, utilizing the composability results above, if timeliness has a latency

requirement less than or equal to λ , RM can transmit requests to each RM_j requiring that $\delta_j + \lambda_j < \lambda/n$. Similarly, if aggregate accuracy was required to be greater than A_0 , then as long as $(1 - \epsilon_j) > \sqrt[n]{A_0}$, relative to each sub-sequence of tasks τ_j , then end-to-end accuracy would be greater than A_0 . Improvements to this scheme should utilize historically validated statistical distributions describing the predictable percentage of total latency required by each logical aggregate task. Alternatively, but much more computationally intensive, RM could request all possible mappings (or a subset thereof, based on additional historical data) from each level-two resource manager; in effect gathering a comprehensive representation of the utility function $U_{App,t}$, where App here represents each aggregate task, and utilize well developed decision making policies at the RM level to complete the process. Realistically, efficient resource management will utilize a combination of the above, together with more refined methodologies. In any event, in order for RM to arrive at a decision regarding the allocation of resources, information must pass to RM from the next layer down, which in turn must receive similar information from managers under its control, etc.

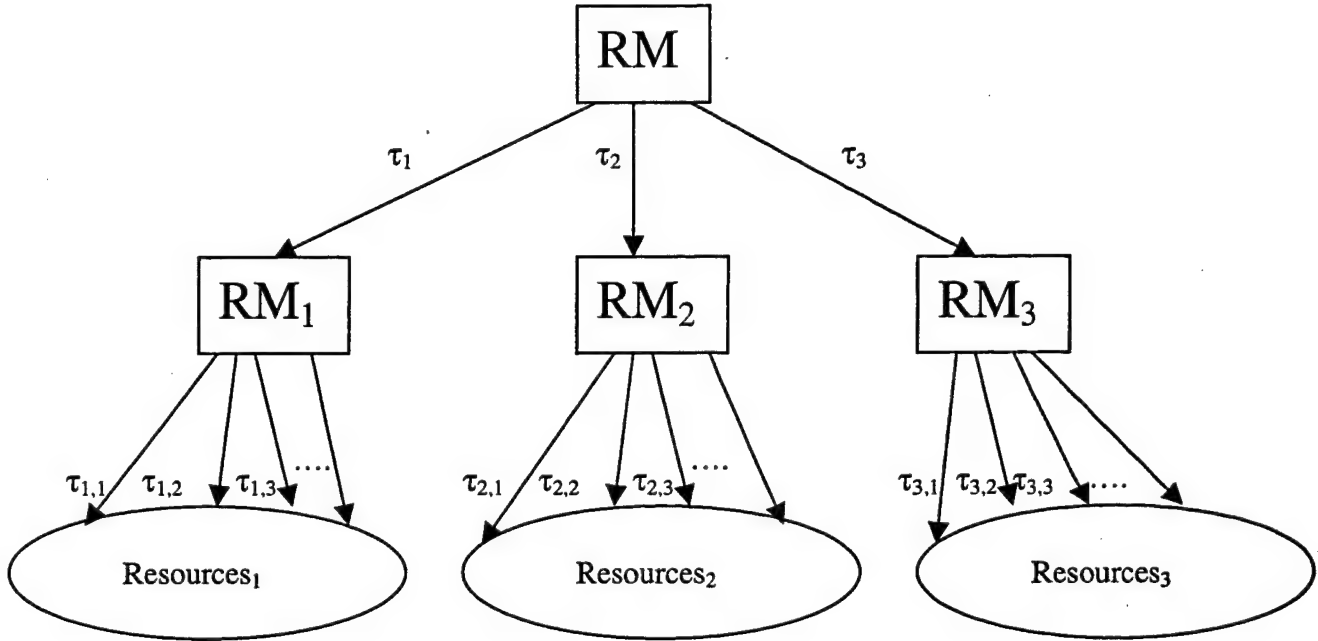


Figure 4

5.9 Topological Structure

Of particular interest with respect to resource management is the need to compare the benefit and cost (in performance) of two apparently different resource mappings towards the implementation of the same application. Mathematically speaking, this can be addressed from a topological perspective. Specifically, based on the quantitative nature of the QoS attributes it is possible to define a metric on the range of RM analogous to

distance between objects in a space. This metric can be defined both at the component level and at the end-to-end flow level in terms of the TPA results and the system 'path' taken to execute the application. Further, this metric is consistent with the QoS attribute values in the sense that two such paths are 'close' only in case they use many of the same resources for the same tasks, and provide similar TPA results.

In particular, suppose S_1 and S_2 are two choices of flows which each yield sub-application A_i . Suppose further that TPA_1 and TPA_2 denote the resulting TPA outputs, respectively. Since each of these triples is a point in Euclidean 3-space, \mathbf{R}^3 , there are numerous choices for distance between them. However, QoS_{etc} , like the benefit function, is not one-to-one in the sense that it is possible to get identical TPA outputs traveling along different paths. That is, it is possible to choose different sub-systems to achieve intermediate tasks and yet arrive at identical TPA values. On the other hand, it is not desirable to consider such solutions equivalent, at least not from a topological point of view, since this representation takes into account only the TPA outputs and not the resources committed to the execution of the application. These resources could possibly be better utilized elsewhere in the system and could play an important role with regards to tradeoffs. As such, the information relative to resource allotment should be reflected in the topology.

Alternatively, let M be an $n \times m$ matrix, where n is the number of tasks, τ_i , and m is the number of sub-systems (or components), s_k , available to execute the n tasks. In terms of the resource management discussion above, and from the perspective of a single resource manager RM_j , m represents the number of resources under the control of RM_j . Let $M = m_{i,j}$, where for each $i < n+1$, $j < m+1$, $m_{i,j}$ is the ordered triple (T_i, P_i, A_i) , namely the TPA output of QoS_i , assuming τ_i is realized through component (or sub-system) j , and is zero otherwise. Here we are assuming that for each i , τ_i is realized through a single sub-system, or single component. From an a posteriori perspective this representation is well defined and is unique for each choice of system resources; that is, for each choice of the sequence of components used to satisfy the sequence of tasks, τ . Further, there are a number of equivalent choices for metrics in the space of $n \times m$ real-valued matrices, each yielding an equivalent topology for the space of flows. More importantly, these choices for metrics, and hence for the resulting topology, are consistent with our idea of "closeness" for two flows. Namely two flows should be close only in case they use primarily the same components with similar (T, P, A) values; and each time they use different components the resulting (T, P, A) entries are small, which implies that only a small portion (relative to TPA) of the overall application was performed by that component. From an a priori perspective each entry in M represents a statistical distribution, or triple thereof, representing expected values for TPA from each component. Relevant choices for metrics can be made in this case also. These vary from simplistic methods based on numerical values for TPA in the form of ordered pairs representing the mean and standard deviation of each distribution, (which reduces to the a

posteriori case) to more sophisticated and computationally intensive, but somewhat standard metrics for comparing continuous distributions.

5.10 Statistical Considerations

As mentioned above, the a priori characteristic of resource management is statistical in nature and is further complicated by the dynamic environment within complex DIS. Typically, each resource manager provides information to the manager one level up, such as, "The probability that latency for logical task τ_j will be less than δ_j is ρ_j ." The problem here is that since most of these tasks are independent, these probabilities are multiplicative. Thus, if App is decomposed into n logical subtasks, each of which is associated with a projected upper bound on latency, say δ_j , each with probability ρ_j , then

the total projected latency for App is bounded above by $\sum_{j=1}^n \delta_j$ with probability $\prod_{j=1}^n \rho_j$.

That is, RM can conclude from the information received from the lower-level managers that the probability that App can be delivered with latency less than $\delta_1 + \delta_2 \dots \delta_n$ is $\rho_1 \rho_2 \dots \rho_n$.

At first glance this appears to be a significant problem, since for large n this product approaches 0 quickly depending on the distance from 1 of each of the ρ_j . For example, if $n = 50$ and each of ρ_j is only 99%, then the probability of delivering the upper bound on latency is only about 60% which of course is unacceptable in most cases. If, on the other hand, the user wants a 90% guarantee, each ρ_j would need to be at least 99.7895%, which might be difficult to guarantee across the system.

Further, this multiplicity of probabilities holds true for each attribute within TPA, not just latency, regardless of the composability features inherent to the particular attribute. For example, composability relative to the volume of data is not additive, as is the case with latency, but is simply the minimum amount of data produced or transported at any one step in the execution process. Nevertheless, the probability of guaranteeing this minimum throughout is again the product of the probabilities from each stage. Further, this multiplicity holds at every level of resource management. Thus, if RM has only three managers one level down with respect to App, resulting in projected attribute values based on the product of only three probabilities (i.e. $n = 3$ at level-two), each of these probabilities in turn would generally result from the product of more probabilities stemming from still lower-level functionality. In other words, there is no way to alleviate this problem by reducing the number of resource managers unless this in turn reduced the number of resources required, and each had probabilities close to 1.

On a more positive note, $n = 50$ is arbitrary and represents a fine-grained decomposition of both App and the associated resources. More importantly, current system performance

in general is much better than these figures indicate, which implies that the majority of the ρ_j in general are very close to 1. Thus, the problem before researchers at this time is the development of a systematic methodology for such decompositions and the identification of the corresponding probability distributions for each QoS attribute.

5.11 Logistical Considerations

There are two intimately intertwined logistical considerations regarding the RM scenario presented above; the volume of information processed and/or transmitted between the levels of management, and the degree of granularity used for the TPA values. For example, suppose App is subdivided into high-level logical tasks $\tau = \tau_1, \tau_2, \dots, \tau_m$ with resource managers RM_1, \dots, RM_m . That is, for simplicity we assume the single task τ_j is executed under the control of RM_j . Thus, it might appear that m pieces of information travel up to RM, each defining the TPA output resulting from the particular resource mapping. On the contrary, each "piece" of information contains at least three quantities, one for each attribute of TPA, assuming incorrectly here that each class of TPA attribute is not further decomposed (e.g. T can be decomposed into latency, jitter, synchronization, etc.) requiring the transmittal of additional information. Further, in order to describe the entire range of possible mappings for resource allotment, which is what is required in order to make full use of the benefit function to make intelligent tradeoffs, the information must indicate a whole range of TPA values. Thus, the question becomes what level of granularity should be used for TPA? Latency, for example, could be broken down into increments as coarse as seconds, or as fine as nano-seconds. Unfortunately, it is not useful to make simple statements such as "resource r can deliver task τ_j with latency less than δ , with probability ρ ", since latency means nothing without precision and accuracy. Thus, combinations of these T, P, and A values must be given in some orderly fashion. Further, if n_T , n_P , and n_A represents the granularity of the levels used for timeliness, precision, and accuracy, respectively, (i.e. there are n_T possible values of timeliness, etc.) then the total number of possible values for TPA for each task is given by the product $n_T n_P n_A = n^*$. The total number of possible combinations of TPA for App is then mn^* .

One method for addressing the logistics associated with this difficulty is illustrated by first taking a very coarse grained approach to the levels of granularity for TPA. Assume for simplicity that the levels for each of T, P, and A are given by either Good (G), Fair (F) or Poor (P), with each of these well defined in a quantitative manner for each application, task, etc. Thus, in this case there are only $3 \times 3 = 27$ possibilities for a characterization of TPA for each task, and hence each resource manager need only check if each of these (or an appropriate subset thereof) can be delivered along with the corresponding probability. These designations for TPA typically look like GGF, or FGF, etc., and are partially ordered by $TPA_1 < TPA_2$ only in case each component of TPA_1 is less than or equal to the

corresponding component of TPA_2 , and they are incomparable otherwise. For example, $GFF < GGF$, but GFF is incomparable with FFG .

From a user perspective, this partial order is consistent with user requirements in the sense that given the choice, if $TPA_1 < TPA_2$ then the user will definitely prefer TPA_2 . However, when points within this partial order are incomparable, additional information from the user and/or the system may be needed to reduce the number of values in the TPA range that need to be assessed. This information should be included in the user profile (via the benefit function) and/or the system policies. Although these levels are given in non-quantified terms (i.e. G, F, or P), clearly it is trivial to store these in numerical form. As a final note, the granularity can be extended to any number of levels, the only drawback of course being the geometric increase in the number of possibilities. This course-grained approach is reasonable for initial modeling and experimentation.

5.12 Global Perspective

Quality of Service thus far in this report has been presented from the perspective of a single user and application as opposed to the global view involving multiple users executing multiple applications simultaneously. Similar to the discussions above, this global view has both a posteriori and a-priori perspectives. However, the authors choose not to elaborate on these perspectives here, assuming their relevance is clear.

5.12.1 Global Benefit Function

In addition to the subjective benefit, B , perceived by each individual user as a result of executing a single application, there is a subjective global benefit perceived by the aggregate community of users resulting from the execution of a set of applications. In a detailed sense, this Global Benefit, denoted $\overline{B}_{\overline{U}, \overline{App}, t}$, is a function, which for each set of users, \overline{U} , and applications, \overline{App} , at each point in time, t , maps sets of TPA values (one TPA triple for each user/application) in D to $[0,1]$. Thus, just as the goal for B is to map each TPA triple to $[0,1]$, the goal for \overline{B} is to map each collection of (User/App, TPA) pairs, representing the aggregate of all TPA triples for all applications executing at time t , to a single value in $[0,1]$.

The values for \overline{B} are highly dependent on the individual benefit functions, B , and on the aggregate profile for the community of users. It is clear that the perceived benefit to the community of users is dependent on the benefits perceived by the individuals within the community in the sense that if all users are satisfied, then the community as a whole is satisfied. It is not clear, however, how this relationship manifests itself in case some users are satisfied and others are not. This is the reason \overline{B} depends also on the aggregate community profile. The community profile in this case refers to the policy(s) by which \overline{B} is determined from the collection of individual benefit functions. For example, keeping in mind that the goal of QoS_{etc} is to maximize user satisfaction as defined by the

benefit function, one such policy might define aggregate benefit as the average of the individual benefits. However, this would imply that the situation in which half the users receive benefit 1, while the other half receives benefit 0, is equivalent to the case where all users receive benefit $\frac{1}{2}$. If these two scenarios are equally desirable by the user community, then this representation for global benefit is appropriate. If, on the other hand, these two are not perceived as equivalent to the user community, then a different definition/profile for global benefit must be used, especially since the global benefit function will be used to make adaptive resource management decisions.

In the case of global benefit, then, the issue of prioritization plays a key role. It is conceivable, for example, that the benefit perceived by the community of users as a whole at a particular point in time may be high provided a single application above all others receives high quality of service. This is particularly true in times of war, when the satisfactory completion of a single high-priority application can mean the difference between loss of life and victory. At different times, however, it might be perceived as higher benefit to the community of users to assure the completion, albeit at lower levels of QoS_{etc} , of the majority of the lower-priority applications rather than a single high-priority task. This type of "weight" attached to each individual benefit is just one of the issues that needs to be incorporated into each user-community profile and then represented in \bar{B} .

5.12.2 Global Utility Function

Just as the benefit function has a global representation, so too does the Utility Function, U . The extension of U to the Global Utility Function, \bar{U} , is much more easily described than the extension of B to \bar{B} . First, recall that $S_{App,t}$ represents the set of possible resources available to execute App at time t . Thus, $S_{App,t}$ is a mapping that for each t assigns to App a set of resources. Thus, we extend $S_{App,t}$ and denote by $\bar{S}_{\bar{App},t}$ the set of all possible mappings from the set of applications \bar{App} to the set of resources available at time t . That is, each application in \bar{App} is executed by a resource in $\bar{S}_{\bar{App},t}$. Given that a single set of resources can be used to [virtually and] simultaneously execute multiple applications, this mapping is not necessarily one-to-one. We can then extend U to \bar{U} in the natural way by assigning to each collection of (application, resources) pairs the resulting collection of points (or distributions) in D , the space of TPA n-tuples/distributions. Thus, QoS_{etc} is naturally extended to Global End-to-End Quality of Service by the definition $\bar{QoS}_{etc} = \bar{B}_{\bar{U},App,t} \circ \bar{U}_{App,t}$.

5.12.3 Global Composability/Algebraic Structure

From the global perspective, Composability theory as described above offers few new results due to the fact that end-to-end QoS for each application still depends on the intermediate TPA values for each task within the single application. Thus, Global Composability must be tied directly to Global Benefit, which is as yet appears to be an

immature concept. However, the intermediate component TPA-outputs are highly dependent on system load and other anomalies. Thus, the representation of each component as a transformation of information under each HW/SW/services/anomaly configuration is a global-level problem that must be solved in order to build a highly developed composability theory.

The Algebraic Structure developed at the individual user/application level and which is dependent on the Composability Theory also extends naturally to the global perspective. Rather than developing fully the rather cumbersome notation required to keep track of the many applications and their associated decompositions into tasks, it suffices to note that the individual algebra was based on cumulative TPA (as defined by composability). This cumulative TPA occurred as the result of the cumulative individual application tasks executed either sequentially or over time. Similarly, in the global case the TPA values accumulate in the aggregate as a result of accumulating for each individual application. Thus, in the continuous case, the aggregate TPA at time $t + j$ is simply the aggregate TPA through time t , composed with the TPA gained (as determined by the Composability Theory applied to each application) from time t to $t + j$. This continuous theory for the global case mimics precisely that for the individual case. The discrete theory also extends in the natural manner, the notation becoming very cumbersome.

5.12.4 Global Adaptive Resource Management

Global Resource Management from an object-definition viewpoint differs little from the resource management discussed above due to the fact that a set of applications, decomposed individually into tasks, yields a single set of tasks, just as in the individual case. However, from a control/decision-making perspective, multiple applications result in limited resources and introduce the possibility for tradeoffs between both users/user-classes and the QoS attributes. Tradeoffs in the individual case are designed to re-allocate resources in order to provide a higher level of QoS_{etc} (TPA values). The fact that multiple combinations of the TPA values represent equivalent individual QoS_{etc} allows for better utilization of system resources, 'trading' between scarce resources without causing negative QoS results. This type of tradeoff utilizes the Benefit Function, which incorporates the fact that it is possible to provide equal benefit to the user by 'trading' one TPA value for another, and the fact that different combinations of resources provide different combinations of TPA. Thus, the question in the individual case becomes: 'which combination of resource allocation yields the highest benefit to the user while satisfying system policy/requirements?'

Tradeoffs at the global level require tradeoffs not only between the TPA attributes of a single user/application, but tradeoffs between the benefits provided to each user. Since global benefit is a function of the individual benefits, albeit that relationship is as-yet undetermined and depends on the particular user community, it is possible to tradeoff benefit between individual users with the goal of increasing global benefit. For example, higher priority users should receive higher assurances/probabilities of application

completion within acceptable QoS_{ete} ranges. It is possible to lower the benefit for a single user/user-group in order to raise the benefit for another user/user group resulting in a higher global benefit. Obviously the concepts of prioritization and preempt-ability play an important role in this process, but only to the degree that these concepts must be well developed and well understood in order to identify a well-defined global benefit function. Once the global benefit function is well defined, the question of tradeoffs at the global level is equivalent again to an identification of all possible such tradeoffs and their resultant effects. This information is then used as a control mechanism to assist RM in resource allocation and re-allocation.

5.13 Summary

The next generation of users requires information systems to reliably provide very specific properties. Delivering these properties requires first their identification and subsequently their implementation. Given the divergent interests of the user communities from which these requirements originate, and the system “engineers” who are responsible for their implementation, a unifying common framework is needed. This is the role of the end-to-end quality of service model. QoS_{ete} is the composite of the Utility function, U , which maps available system resources to statistically deliverable TPA sets, and the Benefit function, B , which then maps TPA to user-perceived benefit. QoS_{ete} provides a measurement mechanism by which intelligent decisions can be made regarding dynamic or static resource allocation or reallocation within a fixed system, and regarding system design and upgrade-ability. The composability theory enabling modular design and/or QoS_{ete} predictability is based on a decomposition of applications into logical tasks, the corresponding execution mapping of these tasks to system resources, and finally the unearthing of the relationship between QoS_{ete} and QoS for the intermediate tasks. The composability theory forms the basis for the algebraic structure of QoS_{ete} . Application “paths” consist of time-sequential mappings of application tasks to the system resources executing the tasks. The resultant QoS for each task then “accumulates” based on the composability theory. This accumulation provides an algebraic structure resembling that found in flow-semigroups. The multi-level resource management mechanism provides dynamic global resource [re] allocation based on the global benefit function. The global resource management process is assisted through a topological structure on the space of application paths. This structure provides a metric-based comparison of the paths in terms of their resultant TPA values, and hence provides a QoS_{ete} -based comparison mechanism. Future research needs to unearth the underlying theory for identifying user requirements, for identifying U and B in terms of TPA or similar basic information attributes, the decomposition of applications and system resources, and the composability thereof.

5.14 Future Directions

The following have been identified as directions that warrant further investigation, **in addition** to continuing research in the areas discussed above:

- Difference equations. The state of the system at each point in time depends on applications currently being executed and those anticipated. Given that resource management decisions are based on the current and expected state of the system, this representation could be very useful. This model reflects a standard type of system of difference equations that describes the change in the system states as the difference: Flow-in minus Flow-out.
- Control Theory. The resource management mechanism, based on intelligent tradeoffs, is a control mechanism designed to change the state of the system. As such, control theory should contribute to this theory.
- Partial ordering of tasks. It was assumed in the majority of the work above that the application tasks are executed sequentially. While this does not appear to be a major obstacle to the resultant composability theory, the fact that tasks are executed in parallel as well as sequentially needs to be considered.
- Further decomposition of TPA. It is clear that each of T, P, and A represents a super-class of information attributes. Further decomposition needs to be identified as it relates directly to user requirements. Then, a composability theory needs to be developed for each basic attribute.
- Statistical nature of QoS_{etc.}. A variety of methods are available to capture the statistical nature of QoS_{etc.}. A basic theory for implementing each of these is required and should include the tradeoffs between more accurate representation and additional load to the system.
- Global Benefit Function. Profiles for user communities as a whole, given in terms of individual profiles need to be developed in order to identify global benefit.

REFERENCES

- [CA96] Campbell, A., Aurrecochea, and L. Hauw, "A Review of QoS Architectures," *Proc. Of the 4th IFIP International workshop on Quality of Service (IWoS '96)*, Paris, March 1996.
- [CSSDL97] Chatterjee, S., J. Sydir, B. Sabata, M. Davis, and T. Lawrence, "Modeling Applications for Adaptive QoS-based Resource Management," IEEE High Assurance Systems Engineering Workshop, Washington, DC, Aug 1997.
- [L97] Lawrence, Thomas F., "The Quality of Service Model and High Assurance," panel position paper for the IEEE High Assurance Systems Engineering Workshop, Bethesda MD, August 1997.
- [SCDSL97] Sabata, B., S. Chatterjee, M. Davis, J. Sydir, and T. Lawrence, "Taxonomy for QoS Specifications," In the Proceedings of the IEEE Computer Society 3rd International Workshop on Object-oriented Real-time Dependable Systems (WORDS '97), Newport Beach, CA, Feb 1997.

[WNCHL97] Wang, W., H. Nguyen, P. Clark, C. Hammond, T. Lawrence, "An Approach to Mapping Multimedia Application QoS to Resources," Technical report done under AFRL prime contract no. F30602-95-C-0299 and in partnership with SRI International. (chammond@sed.stel.com)

Notation

D – the space of TPA values.

$B_{U,App,t} : D \rightarrow [0,1]$ – Subjective benefit to user U , executing application App , at time t .

$B_{U,App,t}^{-1} [b_1, b_2] = \{ TPA \in D : B_{U,App,t}(TPA) \in [b_1, b_2] \}$. That is, B^{-1} is the functional inverse of B .

V – The set of triples (App, S, I) , where App represents the application, S represents the set of all possible SW/HW/Services/Anomaly configurations, and I represents the interval of time in which App is executed.

$S_{App} = \{ SW/HW/Services/Anomaly \text{ configurations that could be tasked, if available, to execute } App \}$

$S_{App,t} = \{ s \in S_{App} : s \text{ is available at time } t \}$

$U_{App,t} : S_{App,t} \rightarrow D$

$QoS_{ete} = B_{U,App,t} \circ U_{App,t}$

A_j – sub-application (App is realized through the sequential execution of sub-applications)

τ -- Sequence representing a decomposition of sub-application A_j .

S_j – system resource responsible for executing task τ_j .

$QoS_j = U_{\tau_j,t}$. That is, the output of QoS_j is that point in D resulting from the execution of task τ_j . QoS_j is also used synonymously as that output.

P_0 – System Policy.

RM – Resource Manager.

***MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)***

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.